MaxPre : An Extended MaxSAT Preprocessor

Korhonen, Tuukka

Springer International Publishing AG
2017

Korhonen , T , Berg , J , Saikko , P & Järvisalo , M 2017 , MaxPre : An Extended MaxSAT Preprocessor . in S Gaspers & T Walsh (eds) , Theory and Applications of Satisfiability þÿ T e s t i n g S A T 2 0 1 7 : 2 0 t h I n t e r n a t i o n a l C o n f e r e n c e M e l b o u r n e , V I C , A u September 1, 2017 Proceedings . Lecture Notes in Computer Science , vol. 10491 , Springer International Publishing AG , Cham , pp. 449-456 , International Conference on Theory and Applications of Satisfiability Testing , Melbourne , Australia , 28/08/2017 . https://doi.org/10.1007/978-3-319-66263-3

http://hdl.handle.net/10138/309054
https://doi.org/10.1007/978-3-319-66263-3_28

acceptedVersion

MaxPre: An Extended MaxSAT Preprocessor*

Tuukka Korhonen, Jeremias Berg, Paul Saikko, and Matti Järvisalo

HIIT, Department of Computer Science, University of Helsinki, Finland
matti.jarvisalo@helsinki.fi

Abstract. We describe MaxPre, an open-source preprocessor for (weighted partial) maximum satisfiability (MaxSAT). MaxPre implements both SAT-based and MaxSAT-specific preprocessing techniques, and offers solution reconstruction, cardinality constraint encoding, and an API for tight integration into SAT-based MaxSAT solvers.

1 Introduction

We describe MaxPre, an open-source preprocessor for (weighted partial) maximum satisfiability (MaxSAT). MaxPre implements a range of well-known and recent SAT-based preprocessing techniques as well as MaxSAT-specific techniques that make use of weights of soft clauses. Furthermore, MaxPre offers solution reconstruction, cardinality constraint encoding, and an API for integration into SAT-based MaxSAT solvers without introducing unnecessary assumptions variables within the SAT solver. In this paper we overview the implemented techniques, implementation-level decisions, and usage of MaxPre, and give a brief overview of its practical potential. The system, implemented in C++, is available in open source under the MIT license via `https://www.cs.helsinki.fi/group/coreo/maxpre/`.

Due to space limitations, we will assume familiarity with conjunctive normal form (CNF) formulas and satisfiability. An instance of (weighted partial) maximum satisfiability (MaxSAT) consists of two CNF formulas, the hard clauses F_h and the soft clauses F_s, and a weight function w associating a non-negative weight with each soft clause. A truth assignment that satisfies the hard clauses is a solution, and is optimal if it minimizes cost, i.e., the weight of the soft clauses left unsatisfied, over all solutions.

The preprocessing flow of MaxPre is illustrated in Fig. 1. Given a MaxSAT instance in DIMACS format (extended with cardinality constraints, see Section 2.1), MaxPre starts by rewriting all cardinality constraints to clauses (step 1). MaxPre then enters the first preprocessing loop (step 2), using only techniques that are directly sound for MaxSAT (see Section 2.2). The sound use of SAT-based preprocessing more generally requires extending each (remaining) soft clause C with a fresh label variable l_C to form $C \vee l_C$ and then restricting the preprocessor from resolving on the added labels [4, 5]. Labelling of the soft clauses with assumption variables is done in steps 3–5. First (step 3) MaxPre applies group detection [7] to identify literals in the input that can be directly used as labels. All soft clauses that remain without a label are then given one (step 4);

* Work supported by Academy of Finland (grants 251170 COIN, 276412, and 284591) and DoCS Doctoral School in Computer Science and Research Funds of the University of Helsinki.

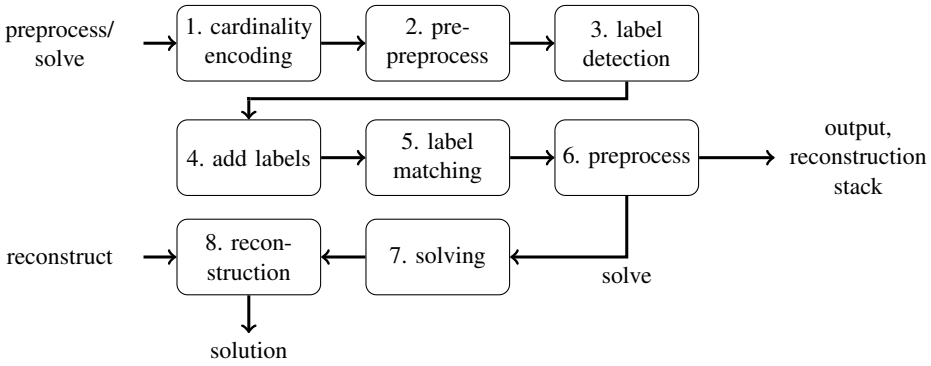

Fig. 1. MaxPre preprocessing flow

during the rest of the preprocessing, all clauses are treated as hard and the weight w_C of a soft clause C is associated with the label l_C attached to C. After each clause is instrumented with a label, a novel technique *label matching* (step 5, see Section 2.2) is used to identify labels which can be substituted with other labels in the instance. The main preprocessing loop (step 6) allows for applying all implemented techniques. Depending on how MaxPre was invoked (see Section 2.3), MaxPre will then either output the resulting preprocessed MaxSAT instance or invoke a given external MaxSAT solver binary on the instance. In the former case, MaxPre will provide a solution reconstruction stack in a separate file. In the latter, MaxPre will internally apply solution reconstruction on the solution output by the solver. We will give more details on the internals, usage, and API of MaxPre, as well as a brief overview of its performance in practice.

2 Supported Techniques

2.1 Cardinality Constraints

MaxPre offers cardinality network [2] based encodings of cardinality constraints, encoded as clauses (Step 1) before preprocessing. This allows the user to specify cardinality constraints over WCNF DIMACS literals as an extension of the standard WCNF MaxSAT input format. Cardinality constraints can be specified using lines of form CARD $l_1\ l_2\ \ldots\ l_n \circ K$, where each l_i is a literal of the formula, $\circ \in \{<, >, <=, >=, =, !=.\}$, and $K \in \mathbb{N}$. The constraints are encoded as hard clauses enforcing $\sum_{i=1}^{n} l_i \circ K$. MaxPre can also encode the truth value of $\sum_{i=1}^{n} l_i \circ K$ to a specific literal L. A line of form CARD $l_1\ l_2\ \ldots\ l_n \circ K$ OUT L is rewritten as $L \rightarrow \sum_{i=1}^{n} l_i \circ K$. Lines of form CARD $l_1\ l_2\ \ldots\ l_n\ \circ\ K$ OUT L IFF extend the implication to an equivalence, and are rewritten as $L \leftrightarrow \sum_{i=1}^{n} l_i \circ K$. Additionally, direct control on the output literals of the cardinality networks can be provided. A line of form CARD $l_1\ l_2\ \ldots\ l_n \square K$ OUT $o_1\ \ldots\ o_K$ is encoded as a K-cardinality network [2] where $o_1, \ldots, o_K$ are the output literals of the network and $\square \in \{<:, >:, ::\}$. If $\square$ is $<:$, the network is encoded so that $\neg o_i \rightarrow \sum_{i=1}^{n} l_i < i$ for any i. If $\square$ is $>:$, the network is encoded so that $o_i \rightarrow \sum_{i=1}^{n} l_i \geq i$; and if $\square$ is $=::$, the network is encoded so that $o_i \leftrightarrow \sum_{i=1}^{n} l_i \geq i$.

2.2 Preprocessing

SAT-Based Preprocessing. In addition to removing tautologies, soft clauses with 0 weight and duplicate clauses, MaxPre implements the following SAT-based techniques: unit propagation, bounded variable elimination (BVE) [11], subsumption elimination (SE), self-subsuming resolution (SSR) [11, 13, 17], blocked clause elimination (BCE) [15], Unhiding [14] (including equivalent literal substitution [1, 24, 18, 9]), and bounded variable addition (BVA) [20]. In terms of implementation details, SE and SSR use three different approaches depending on the number of clauses they have to process. The asymptotically worst approach of checking all pairs of clauses can be improved by computing hashes of clauses, and by the asymptotically best approach using the AMS-Lex algorithm [3], implemented in MaxPre based on [21]. The average time complexity of AMS-Lex in finding subsumed clauses seems to be nearly linear (dominated by sorting). We have observed that in practice the BVA implementation can be in cases significantly faster than the implementation given in the original paper; this is achieved by using polynomial hashes of clauses. In contrast to using time stamping (directed spanning trees) as in the original work on Unhiding [14], MaxPre implements Unhiding using undirected spanning trees, which can be provably more effective in terms of the covered binary implications.

MaxSAT-Specific Techniques. MaxPre also includes the following (to the best of our knowledge unique) combination of MaxSAT-specific techniques that work directly on the label variables (recall Fig. 1). The first two are techniques meant to decrease the total number of fresh label variables that are introduced into the formula. Assume MaxPre is invoked on an instance $F = (F_h, F_s, w)$.

- **Group detection** [7] (Step 3) Any literal l for which $(\neg l) \in F_s$, $l \notin C$ for any $C \in (F_s \setminus \{(\neg l)\})$ and $\neg l \notin C$ for any $C \in F_h \cup (F_s \setminus \{(\neg l)\})$ can be directly used as a label [7].
- **Label matching** (Step 5) Label l_C is *matched*, i.e., substituted, with l_D if (i) $w_C = w_D$, (ii) l_C only appears in a single soft clause C, (iii) l_D only appears in a single soft clause D, and (iv) $C \vee D$ is a tautology. MaxPre implements label matching by computing a maximal matching using a standard greedy algorithm.
- **Group-subsumed label elimination (GSLE)** Generalizing SLE [8], label l_D is subsumed by a group of labels $L = \{l_{C_1}, \ldots, l_{C_n}\}$ if for some $l_{C_i} \in L$, we have $l_{C_i} \in C$ whenever $l_D \in C$, and $\sum_{l_{C_i} \in L} w_{C_i} \leq w_D$. GSLE removes group-subsumed labels. SLE corresponds to GSLE with $n = 1$. Since GSLE corresponds to the NP-complete hitting set problem, MaxPre implements an approximate GSLE via a slightly modified version of a classical $ln(n)$–approximation algorithm for the hitting set problem [10, 23].
- **Binary core removal (BCR)** is the MaxSAT-equivalent of Gimpel's reduction rule for the binate covering problem [12]. Assume labels l_C, l_D with $w_C = w_D$ and the clause $(l_C \vee l_D)$. Let $F_{l_C} = \{C_i \mid l_C \in C_i\}$ and assume that (i) $F_{l_C} \cap F_{l_D} = \{(l_C \vee l_D)\}$, (ii) $|F_{l_C}| > 1$, and (iii) $|F_{l_D}| > 1$. BCR replaces the clauses in $F_{l_C} \cup F_{l_D}$ with the non-tautological clauses in $\{(C_i \vee D_j) \setminus \{l_C\} \mid C_i \in F_{l_C} \setminus (l_C \vee l_D), D_j \in F_{l_D} \setminus (l_C \vee l_D)\}$. MaxPre applies BCR whenever the total number of clauses in the formula does not increase. Notice that given l_C, l_D with $w_C = w_D$ and a clause $(l_C \vee l_D)$, assumptions (i)-(iii) for BCR follow by applying SE and SLE.

Structure-based labeling Given a label l and a clause C s.t. C is blocked (in terms of BCE) when assuming l to true, structure-based labelling replaces C by $C \vee l$. The correctness of structure-based labelling is based on the invariant that a clause C is redundant whenever l is true.

2.3 Options and Usage

MaxPre is called from the command line. Full details on command line options are available via `./maxpre -h`. One of the directives `preprocess`, `solve`, `reconstruct` needs to be specified after the input file. `preprocess` and `solve` assume a single input file containing a WCNF MaxSAT instance (possibly with cardinality constraints). Further, `solve` expects the solver binary and its command line arguments to be given via the `-solver` and `-solverflags` options. `reconstruct` expects as input a solution to a WCNF MaxSAT instance and the corresponding reconstruction stack file.

Solution Reconstruction. To map a solution of a preprocessed instance to a solution of the original instance, the reconstruction stack file produced by MaxPre needs to be specified via `-mapfile`. For example, to obtain an original solution from a solution `sol0.sol` of the preprocessed instances using the reconstruction stack in `input.map`, use `./maxpre sol0.sol reconstruct -mapfile=input.map`. To obtain the mapfile when preprocessing, `-mapfile` should be used in conjunction with `preprocess`.

Specifying Preprocessing Techniques. Following [19], MaxPre allows for specifying the order in which individual preprocessing techniques are applied via a technique string The default application order is specified by the string $[bu]\#[buvsrgc]$, i.e. to run BCE and UP in the first preprocessing loop and BCE, UP, BVE, SSR, GSLE and BCR in the second.

Enforcing Time Limits and Bounds. The running time of MaxPre is limited using the `-timelimit` option: e.g., `-timelimit=60` sets the time limit to 60 seconds. This limits the running time of preprocessing techniques somewhat independently of each other. Each technique gets allocated a proportion of the time to use, and if a technique leaves some of its time unused, it is dynamically reallocated to other techniques. By default no time limits are imposed. Another way to prevent MaxPre from wasting efforts on the potentially more time-consuming techniques such as BVE via `-skiptechnique`: e.g., with `-skiptechnique=100`, MaxPre first tries to apply each of such preprocessing technique to 100 random variables/literals in a touched list. If no simplifications occur, MaxPre will subsequently skip the particular preprocessing technique. By default `-skiptechnique` is not enforced.

2.4 API

MaxPre is implemented in a modular fashion, and offers an API for tight integration with MaxSAT solvers. Via the API, the solver becomes aware of labels used for preprocessing which can be directly used as assumptions in SAT-based MaxSAT solving;

without this, the solvers will add a completely redundant new layer of assumption variables to the soft clauses before search [6]. Unnecessary file I/O is also avoided.

To use MaxPre via its API, create a `PreprocessorInterface` object for the MaxSAT instance, call `preprocess` to preprocess, and then `getInstance` to obtain the preprocessed instance. After solving, an original solution is reconstructed by calling `reconstruct`. `PreprocessorInterface` encapsulates the preprocessing trace (mapfile) and maps/unmaps variables to/from internal preprocessor variable indexing. For more concreteness, `main.cpp` implements MaxPre using these API methods, and serves as an example of their use. The API handles literals as `int`-types and clauses as C++ standard library vectors of literals. The most central parts of the API are the following.

`PreprocessorInterface` constructs a `PreprocessorInterface` object from a vector of clauses, a vector of their weights and a top weight, which is used to identify hard clauses.
`preprocess` takes a technique string and preprocesses the instance using given techniques.
`getInstance` returns (by reference) the preprocessed instance as vectors of clauses, weights, and labels.
`reconstruct` takes a model for the preprocessed instance and returns a model for the original instance.
`setSkipTechnique` corresponds to `-skiptechnique` command line flag.
`print*` methods print solutions, instances, mapfiles, or logs to an output stream.

3 Experiments

While an extended empirical evaluation of different components of MaxPre is impossible within this system description, we shortly discuss the potential of MaxPre in practice. For the experiments, we used all of the 5425 partial and weighted partial MaxSAT instances collected and made available by the 2008–2016 MaxSAT Evaluations. Figure 2 show a comparison of MaxPre and the Coprocessor 2.0 [19] preprocessor in terms of the number of variables and clauses, and the sums of weights of the soft clauses,

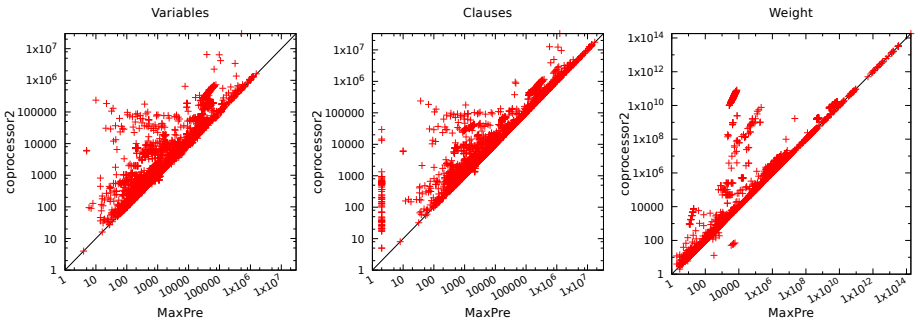

Fig. 2. Comparison of MaxPre and Coprocessor in terms of preprocessing effects: number of clauses and variables, and the sum of the weights of soft clauses, in the preprocessed instances.

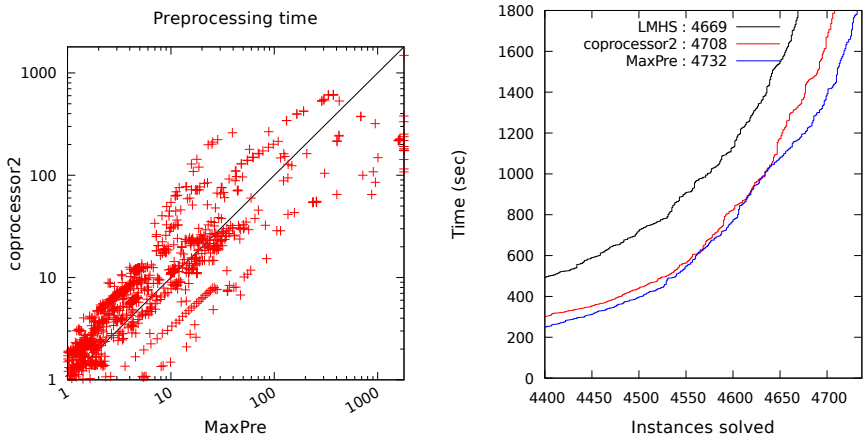

Fig. 3. Left: Comparison of preprocessing times: MaxPre vs Coprocessor. Right: Performance impact on LMHS.

in the output instance. Here we used Coprocessor by first adding a label to each soft clauses, and used the "white listing" option of Coprocessor on the labels to maintain soundness of preprocessing for MaxSAT. The SAT-based techniques used with Coprocessor and MaxPre are the same; the preprocessing loop used in coprocessor consisted of BVE, pure literal elimination, UP, SE, SSR and BCE. MaxPre additionally applied the MaxSAT-specific techniques shortly described earlier in this paper; the first preprocessing loop used BCE and UP and the second BCE (which also removes pure literals), UP, BVE, SE, SSR, GSLE and BCR. MaxPre provides noticeably more simplifications in terms of these parameters, while the preprocessing times as comparable (see Fig. 3 left); in fact, MaxPre performs faster on a majority of the instances. The few timeouts observed for MaxPre are on very large instances ($>$ 10 M clauses) which are in fact not solved by current state-of-the-art MaxSAT solvers. Coprocessor contains fixed constants which switch off preprocessing on very large instances; while we did not enforce any limits on MaxPre for this experiment, the `-timelimit` and `-skiptechniques` options would enable faster preprocessing on such very large intances. In terms of potential impact on solver performance, our LMHS SAT-IP MaxSAT solver [22] previously integrated Coprocessor 2.0 as its internal preprocessor. Figure 3 right shows that, although the effects are mild, replacing Coprocessor with MaxPre in LMHS improves its performance.

4 Conclusions

We introduced MaxPre, an open-source MaxSAT preprocessor with extended capabilities, including cardinality constraint encodings and MaxSAT-specific simplification techniques not implemented in the current (Max)SAT preprocessors. The API of MaxPre allows for tight integration with MaxSAT solvers, e.g., avoiding unnecessary introduction of assumption variables after preprocessing, and potentially opening up further avenues for inprocessing MaxSAT solving [16]. Empirical results suggest that MaxPre is a viable option for integrating preprocessing into MaxSAT solvers.

References

1. Aho, A., Garey, M., Ullman, J.: The transitive reduction of a directed graph. SIAM Journal on Computing 1(2), 131–137 (1972)
2. Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality networks: a theoretical and empirical study. Constraints 16(2), 195–221 (2011)
3. Bayardo, R.J., Panda, B.: Fast algorithms for finding extremal sets. In: Proc. SDM. pp. 25–34. SIAM / Omnipress (2011)
4. Belov, A., Järvisalo, M., Marques-Silva, J.: Formula preprocessing in MUS extraction. In: Proc. TACAS. Lecture Notes in Computer Science, vol. 7795, pp. 108–123. Springer (2013)
5. Belov, A., Morgado, A., Marques-Silva, J.: SAT-based preprocessing for MaxSAT. In: Proc. LPAR-19. Lecture Notes in Computer Science, vol. 8312, pp. 96–111. Springer (2013)
6. Berg, J., Saikko, P., Järvisalo, M.: Improving the effectiveness of SAT-based preprocessing for MaxSAT. In: Proc. IJCAI. pp. 239–245. AAAI Press (2015)
7. Berg, J., Saikko, P., Järvisalo, M.: Re-using auxiliary variables for MaxSAT preprocessing. In: Proc. ICTAI. pp. 813–820. IEEE Computer Society (2015)
8. Berg, J., Saikko, P., Järvisalo, M.: Subsumed label elimination for maximum satisfiability. In: Proc. ECAI. Frontiers in Artificial Intelligence and Applications, vol. 285, pp. 630–638. IOS Press (2016)
9. Brafman, R.: A simplifier for propositional formulas with many binary clauses. IEEE Transactions on Systems, Man, and Cybernetics, Part B 34(1), 52–59 (2004)
10. Chvatal, V.: A greedy heuristic for the set-covering problem. Mathematics of Operations Research 4(3), 233–235 (1979)
11. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Proc. SAT. Lecture Notes in Computer Science, vol. 3569, pp. 61–75. Springer (2005)
12. Gimpel, J.F.: A reduction technique for prime implicant tables. In: Proc. SWCT. pp. 183–191. IEEE Computer Society (1964)
13. Groote, J., Warners, J.: The propositional formula checker HeerHugo. Journal of Automated Reasoning 24(1/2), 101–125 (2000)
14. Heule, M., Järvisalo, M., Biere, A.: Efficient CNF simplification based on binary implication graphs. In: Proc. SAT. Lecture Notes in Computer Science, vol. 6695, pp. 201–215 (2011)
15. Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In: Proc. TACAS. Lecture Notes in Computer Science, vol. 6015, pp. 129–144. Springer (2010)
16. Järvisalo, M., Heule, M., Biere, A.: Inprocessing rules. In: Proc. IJCAR. Lecture Notes in Computer Science, vol. 7364, pp. 355–370. Springer (2012)
17. Korovin, K.: iProver – an instantiation-based theorem prover for first-order logic. In: Proc. IJCAI. Lecture Notes in Computer Science, vol. 5195, pp. 292–298. Springer (2008)
18. Li, C.: Integrating equivalency reasoning into Davis-Putnam procedure. In: Proc. AAAI. pp. 291–296 (2000)
19. Manthey, N.: Coprocessor 2.0 - A flexible CNF simplifier. In: Proc. SAT. Lecture Notes in Computer Science, vol. 7317, pp. 436–441. Springer (2012)
20. Manthey, N., Heule, M., Biere, A.: Automated reencoding of boolean formulas. In: Proc. HVC Revised Selected Papers. Lecture Notes in Computer Science, vol. 7857, pp. 102–117. Springer (2012)
21. Marinov, M., Nash, N., Gregg, D.: Practical algorithms for finding extremal sets. Journal of Experimental Algorithmics 21, article 1.9 (2016)
22. Saikko, P., Berg, J., Järvisalo, M.: LMHS: A SAT-IP hybrid MaxSAT solver. In: Proc. SAT. Lecture Notes in Computer Science, vol. 9710, pp. 539–546. Springer (2016)
23. Slavík, P.: A tight analysis of the greedy algorithm for set cover. In: Proc. STOC. pp. 435–441. ACM (1996)

24. Van Gelder, A.: Toward leaner binary-clause reasoning in a satisfiability solver. Annals of Mathematics and Artificial Intelligence 43(1), 239–253 (2005)